|  |  |
| --- | --- |
| **数据结构与算法 综合训练报告** | |
| 图片包含 标牌  已生成极高可信度的说明 | |
|  | |
|  | |
| **姓名** |  |
| **班级** |  |
| **学号** |  |
| **电话** |  |
| **Email** |  |
| **日期** |  |

# 用图解决问题

**题目背景 1**

**任务1 2**

题目 2

解答 2

**任务2 2**

题目 2

数据设计 2

算法设计 3

主干代码说明 4

运行结果展示 5

总结和收获 5

**任务3 6**

题目 6

数据设计 6

算法设计 6

主干代码说明 7

运行结果展示 7

总结和收获 8

**任务4 8**

题目 8

解答 9

**任务5 9**

题目 9

解答 9

**源代码 9**

# 题目背景

# 任务1

### 一、题目

建立为实现该游戏的图的抽象描述结构，包括图中顶点的意义以及存储的信息、边的意义以及存储的信息。并给出该图的逻辑示意图。

### 二、解答

为实现该游戏的图中，顶点代表演员，边是无向边，代表两名演员共同参演的电影。该图中的数据较为稀疏，可以采取邻接表的方式存储，邻接表中的每一个链表存储一个结点和结点所连接的边。结点借用边的类进行存储，存储一个字符串，即演员的姓名。边的类存储一个字符串，即电影的名称和边指向的结点的下标。

该图的存储逻辑如下：

|  |  |  |  |
| --- | --- | --- | --- |
| “Actor 1”,0🡪 | “Movie 12”,1🡪 | “Movie 13”,2🡪 | ... |
| “Actor 2”,1🡪 | “Movie 12”,0🡪 | ... |  |
| “Actor 3”,2🡪 | “Movie 13”,0🡪 | ... |  |

每一行是一个链表，整个图是链表的数组。

# 任务2

### 一、题目

在任务1的基础上，并结合教材中图的抽象数据类型的定义，设计并实现一个为该游戏而使用的具体的 Graph Class。

### 二、数据设计

考虑Simple.txt的数据规模和复杂度，该图宜使用邻接表方法进行存储。类Graph中应含有一个链表的数组，每一个链表存储着一个顶点和这个顶点所连接的边。类Graph中还应含有当前图的大小和图中的顶点数，以便进行添加和读取操作。

类Edge代表图中的一条从n1指向n2的边。由于Edge类被存储在n1对应的链表中，可以只存储它对应的n2的下标。同时，该类还应存储这条边上的电影名称。在每个链表的第一位，借用Edge类存储图的一个顶点，存储顶点的下标和顶点的名称。

在寻找结点关系的BST算法中还需要利用队列和ArrayList，本Graph类使用了链表、队列、数组三种数据结构。

### 三、算法设计

仅为实现这个游戏无需实现图的所有方法，只需实现插入顶点、插入边的相关操作即可。寻找演员之间的关系即寻找两特定结点之间的最短路径，由于本图的边都是无权的，可以采用广度优先搜索BFS的思想来寻找最短路径。类Graph的主要方法如下：

1. insertEdge：由于边中连接的点以下标形式存储，在插入图之前要先将输入的字符串转化为下标。实现nodeNum方法寻找字符串对应的下标，若不能找到该字符串，则为该字符串新建一个链表，并初始化链表的首位用来存储它。此后，分别对两个顶点对应的链表调用insert方法，给链表中添加一个新的边。
2. nodeNum,nodeString：分别是寻找字符串对应顶点的下标和寻找下标对应的顶点的字符串。
3. findRelations：找到两个字符串对应的顶点之间的关系，返回一个ArrayList<Edge>。先将传入的字符串转换为下标，之后定义新队列用于广度优先搜索、定义boolean数组visited存储结点的访问状态、定义Edge数组father存储每个结点是从哪条边被遍历的。将n1对应链表的第一位加入队列，随后在列表非空的情况下执行循环。若找到了n2，则利用数组father回溯搜索过程，将经过的边逐个加入结果数组中，返回结果数组。若没有找到，若有未被访问的该结点的边指向的下一结点，将该边加入队列，将下一结点对应的father修改为本边，并修改下一结点的访问状态。

### 四、主干代码说明

**public void** insertEdge(String n1,String n2,String str){*//在n1与n2之间插入字段为str的边* **int** num1=nodeNum(n1);  
 **int** num2=nodeNum(n2);  
 **graph**[num1].insert(**new** Edge(num2,str));*//为n1的链表插入边* **graph**[num2].insert(**new** Edge(num1,str));*//为n2的链表插入边*}

以上是插入一条边的方法，它依赖于nodeNum方法，即获得字符串对应的结点下标。

**private int** nodeNum(String str){*//如果str在图中，则返回它的编号，否则新建一个顶点和它的邻接链表* **for**(**int** i=0;i<**num**;i++) {  
 **graph**[i].gotoBeginning();  
 **if** (**graph**[i].getCursor().**str**.equals(str)) {*//每个链表的第一位存储了一个结点，如果这个结点值和str相等* **return** i;  
 }  
 }  
 *//str不在当前图中，为它新建一个链表* **graph**[**num**]= **new** LinkList<>();  
 **graph**[**num**].insert(**new** Edge(**num**,str));  
 **num**++;  
 **return num**-1;  
}  
**public** String nodeString(**int** num){  
 **graph**[num].gotoBeginning();  
 **return graph**[num].getCursor().**str**;  
}

以上是顶点的字符与下标互转的两个方法。

**public** ArrayList<Edge> findRelations(String str1,String str2){*//从n1开始进行广度优先搜索，直到找到n2，返回的是n2到n1的路径* **int** n1=nodeNum(str1),n2=nodeNum(str2);  
 **boolean**[] visited=**new boolean**[**n**];  
 Edge[] father=**new** Edge[**n**];*//用于存储每个被遍历顶点的上级* ArrayList<Edge> relations=**new** ArrayList<>();  
 ResizingQueue<Edge> queue=**new** ResizingQueue<>();  
 **graph**[n1].gotoBeginning();  
 queue.enqueue(**graph**[n1].getCursor());  
 visited[n1]=**true**;  
 father[n1]=**null**;  
 **while** (queue.size()!=0){  
 Edge now = queue.dequeue();  
 **if**(now.**n2**==n2){  
 **while** (now!=**null**){*//回溯遍历过程，写入relations* relations.add(now);  
 now=father[now.**n2**];  
 }  
 **return** relations;  
 }  
 **for** (**graph**[now.**n2**].gotoBeginning();**graph**[now.**n2**].gotoNext();){  
 Edge next=**graph**[now.**n2**].getCursor();  
 **if**(!visited[next.**n2**]){*//next未被遍历* father[next.**n2**]=now;  
 visited[next.**n2**]=**true**;  
 queue.enqueue(next);  
 }  
 }  
 }  
 **return null**;  
}

以上findRelations方法是整个项目的核心。它采用广度优先搜索BFS的思想，如果当前边指向的结点不是要找的结点，就将结点连接到的边加入队列。只要队列不为空，就不断地取出队首进行处理，直到找到要找的结点为止。为在找到结点后能够得知找到的路径，采取一个数组来存储每一个结点是如何被搜索到的，存下结点被访问时通过的边。

### 五、运行结果展示

在下一任务中可以看到本任务实现的图的有效应用。

### 六、总结和收获

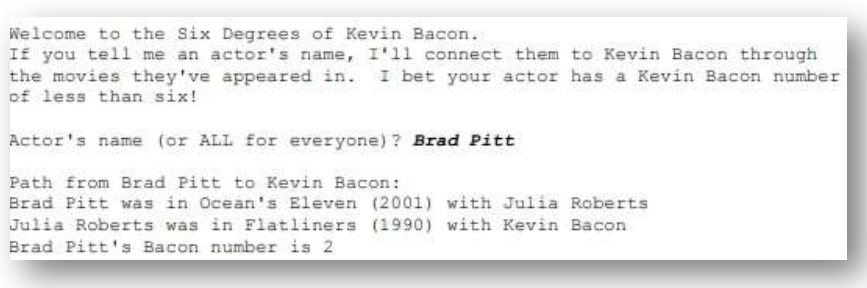
本任务中利用邻接表方法实现了一个图。虽然没有实现图的ADT中的全部方法，但已足够本项目进行使用。实践练习了对图的广度优先搜索算法的实现，加深了对图的组织结构的认识。

# 任务3

### 一、题目

通过给定的数据文件 simple.txt 构建图。Simple.txt 中的格式为：每一行代表一个电影，每行中的信息都用‘/’进行分割，其中第一个信息为电影名称，其后所有的信息都是出现在该电影的演员名。

利用图和相应的算法，你可以根据用户输入的演员名，给出该演员的 Bacon Number，并且列出该数计算的依据，也就是通过哪些电影建立了和 Kevin Bacon 的联系。运行的样例模式如下：



### 二、数据设计

本任务中需要通过读取Simple.txt的数据构建一个图，并调用图的findRelations方法，找到输入的演员与Kevin Bacon之间的关系。

### 三、算法设计

定义类KevinBacon，在其中实现main方法。利用Scanner读取文件的每一行，并将其以分隔符/分隔，数组的第一位是电影的名称，后几位是演员的姓名。对于每两位演员，调用一次insertEdge方法，构建出一个适用于此任务的图。获取用户的输入并调用findRelations方法，对于返回的ArrayList，其最后一位是开始寻找时的结点，不是一条边，则遍历除此位外的所有位，根据边中的信息输出对应的字段。最后，演员的Bacon Number应为数组的长度减一，输出对应的字段。

### 四、主干代码说明

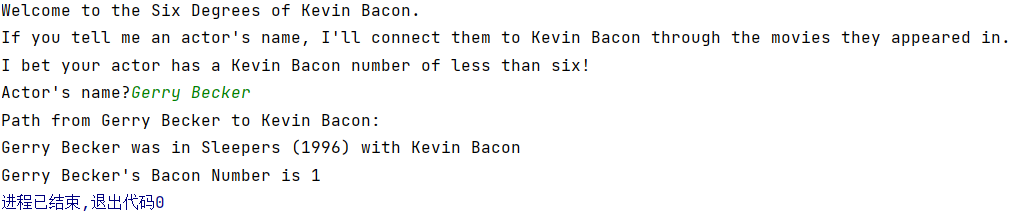
**while** (scanner.hasNextLine()){*//建立Graph* String[] input=scanner.nextLine().split(**"/"**);  
 **for** (**int** i=1;i< input.**length**-1;i++){  
 **for**(**int** j=i+1;j<input.**length**;j++){*//为输入人名中的每一对人名添加对应的边* graph.insertEdge(input[i],input[j],input[0]);  
 }  
 }  
}

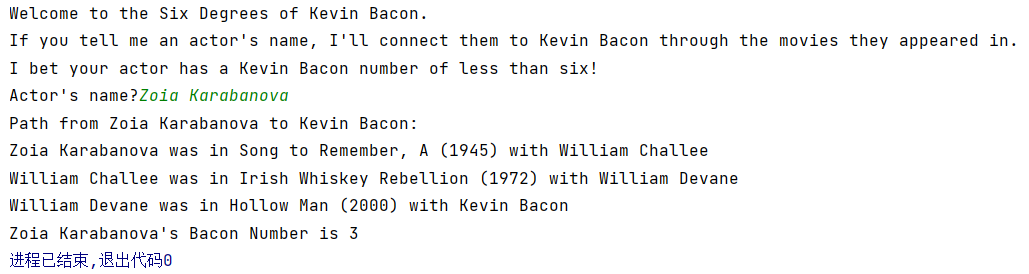
以上是根据文件建立图的相关代码。

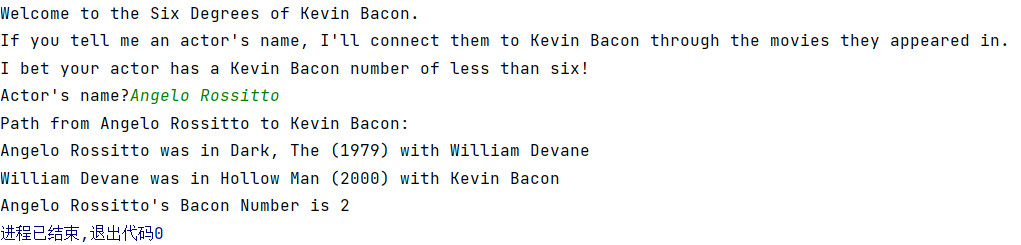
ArrayList<Edge> relation=graph.findRelations(actor1,actor2);  
**if**(relation==**null**||relation.size()==0){*//找不到对应的关系* System.***out***.print(**"Your input is invalid!"**);  
 **return**;  
}  
System.***out***.printf(**"Path from %s to %s:\n"**,actor1,actor2);  
**for**(**int** i=relation.size()-2;i>=0;i--){*//列表最后一位不是关系* actor2=graph.nodeString(relation.get(i).**n2**);  
 System.***out***.printf(**"%s was in %s with %s\n"**,actor1,relation.get(i).**str**,actor2);  
 actor1=actor2;  
}  
System.***out***.printf(**"%s's Bacon Number is %d"**,actor,relation.size()-1);

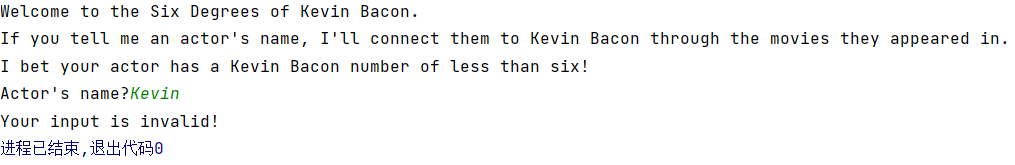
以上是根据输入找到与Kevin Bacon的关系并进行相关处理和输出的相关代码。

### 五、运行结果展示









### 六、总结和收获

本任务实现了一个基本的“Six Degrees of Kevin Bacon”游戏。依赖于任务2中实现的用于此游戏的图，本游戏能够正常运行。本任务体现了图在实际问题中的重要应用。

# 任务4（选做）

### 一、题目

Complex.txt 是一个数据规模远超 Simple.txt 的文件，两者的格式是一样的。尝试使用任务 3 的解决方案执行 Complex.txt。如果运行效果不理想，请分析问题的原因，并尝试进行改变。

### 二、解答

经过尝试，在根据Complex.txt构建图时所耗费的时间很长。这可能与图的存储结构、寻找结点关系的算法相关，可以从这些方面予以改进。

# 任务5

### 一、题目

在你的日常学习生活中，寻找一个可以用图解决的问题原型，描述该问题原型，并陈述如何将该问题原型抽象成图的表示。

### 二、解答

问题原型：实现从学校内的一点到另一点的路线规划和导航。

图的表示：每条道路的路口抽象为一个结点，每条道路抽象为一个边。根据学校的地图构建一个图，图中的边的权等于这条道路的长度，此后运用Dijkstra算法寻找结点之间的最短路径。

### 附录：源代码

##### Graph.java

**import** java.util.ArrayList;  
  
**public class** Graph {  
 *//采取邻接表的存储方式* **private** LinkList<Edge>[] **graph**;*//边构成的链表的数组，第一位存储一个顶点* **int num**=0;*//顶点的个数* **int n**;*//图的大小（能够存储的顶点个数）* **public** Graph(**int** n){  
 **this**.**n**=n;  
 **graph**=**new** LinkList[n];  
 }  
 **public void** insertEdge(String n1,String n2,String str){*//在n1与n2之间插入字段为str的边* **int** num1=nodeNum(n1);  
 **int** num2=nodeNum(n2);  
 **graph**[num1].insert(**new** Edge(num2,str));*//为n1的链表插入边* **graph**[num2].insert(**new** Edge(num1,str));*//为n2的链表插入边* }  
 **private int** nodeNum(String str){*//如果str在图中，则返回它的编号，否则新建一个顶点和它的邻接链表* **for**(**int** i=0;i<**num**;i++) {  
 **graph**[i].gotoBeginning();  
 **if** (**graph**[i].getCursor().**str**.equals(str)) {*//每个链表的第一位存储了一个结点，如果这个结点值和str相等* **return** i;  
 }  
 }  
 *//str不在当前图中，为它新建一个链表* **graph**[**num**]= **new** LinkList<>();  
 **graph**[**num**].insert(**new** Edge(**num**,str));  
 **num**++;  
 **return num**-1;  
 }  
 **public** String nodeString(**int** num){  
 **graph**[num].gotoBeginning();  
 **return graph**[num].getCursor().**str**;  
 }  
 **public** ArrayList<Edge> findRelations(String str1,String str2){*//从n1开始进行广度优先搜索，直到找到n2，返回的是n2到n1的路径* **int** n1=nodeNum(str1),n2=nodeNum(str2);  
 **boolean**[] visited=**new boolean**[**n**];  
 Edge[] father=**new** Edge[**n**];*//用于存储每个被遍历顶点的上级* ArrayList<Edge> relations=**new** ArrayList<>();  
 ResizingQueue<Edge> queue=**new** ResizingQueue<>();  
 **graph**[n1].gotoBeginning();  
 queue.enqueue(**graph**[n1].getCursor());  
 visited[n1]=**true**;  
 father[n1]=**null**;  
 **while** (queue.size()!=0){  
 Edge now = queue.dequeue();  
 **if**(now.**n2**==n2){  
 **while** (now!=**null**){*//回溯遍历过程，写入relations* relations.add(now);  
 now=father[now.**n2**];  
 }  
 **return** relations;  
 }  
 **for** (**graph**[now.**n2**].gotoBeginning();**graph**[now.**n2**].gotoNext();){  
 Edge next=**graph**[now.**n2**].getCursor();  
 **if**(!visited[next.**n2**]){*//next未被遍历* father[next.**n2**]=now;  
 visited[next.**n2**]=**true**;  
 queue.enqueue(next);  
 }  
 }  
 }  
 **return null**;  
 }  
}

##### Edge.java

**public class** Edge {  
 *//代表一条边* **public int n2**;*//这条边连接到的位置* **public** String **str**;*//这条边上的值，即电影的名字* **public** Edge(**int** n2,String str){  
 **this**.**n2**=n2;  
 **this**.**str**=str;  
 }  
}

##### KevinBacon.java

**import** java.io.File;  
**import** java.io.FileNotFoundException;  
**import** java.util.ArrayList;  
**import** java.util.Scanner;  
  
**public class** KevinBacon {  
 **public static void** main(String[] args) **throws** FileNotFoundException {  
 Graph graph=**new** Graph(50000);  
 Scanner scanner=**new** Scanner(**new** File(**"Simple.txt"**));  
 **while** (scanner.hasNextLine()){*//建立Graph* String[] input=scanner.nextLine().split(**"/"**);  
 **for** (**int** i=1;i< input.**length**-1;i++){  
 **for**(**int** j=i+1;j<input.**length**;j++){*//为输入人名中的每一对人名添加对应的边* graph.insertEdge(input[i],input[j],input[0]);  
 }  
 }  
 }  
 System.***out***.println(**"Welcome to the Six Degrees of Kevin Bacon."**);  
 System.***out***.println(**"If you tell me an actor's name, I'll connect them to Kevin Bacon through the movies they appeared in."**);  
 System.***out***.println(**"I bet your actor has a Kevin Bacon number of less than six!"**);  
 Scanner scannerin=**new** Scanner(System.***in***);  
 System.***out***.print(**"Actor's name?"**);  
 String actor=scannerin.nextLine();  
 String actor1=actor;  
 String actor2=**"Kevin Bacon"**;  
 ArrayList<Edge> relation=graph.findRelations(actor1,actor2);  
 **if**(relation==**null**||relation.size()==0){*//找不到对应的关系* System.***out***.print(**"Your input is invalid!"**);  
 **return**;  
 }  
 System.***out***.printf(**"Path from %s to %s:\n"**,actor1,actor2);  
 **for**(**int** i=relation.size()-2;i>=0;i--){*//列表最后一位不是关系* actor2=graph.nodeString(relation.get(i).**n2**);  
 System.***out***.printf(**"%s was in %s with %s\n"**,actor1,relation.get(i).**str**,actor2);  
 actor1=actor2;  
 }  
 System.***out***.printf(**"%s's Bacon Number is %d"**,actor,relation.size()-1);  
 }  
}

本项目还用到了先前实验中实现的队列和链表，在此不再列举其源代码。